MAPDE: MongeAmpere Partial Differential Equation

Tianma Shen, Mingjia Xue, Xinyue Wang, Qiming Yuan

Santa Clara University

May 31, 2021

Overview

MongeAmpere Partial Differential Equation

- A nonlinear second-order partial differential equation
- Frequently arise in differential geometry
- Cannot get analytic solutions (Dimension ¿ 2)

$$\det\left(D^2 u(x)\right) = \frac{\rho_X(x)}{\rho_Y(\nabla u(x))}, \quad \text{for } x \in X \tag{1}$$

where ρ_X is a probability density supported on X and ρ_Y is a probability density supported on Y. MAPDE subjects to the transport condition(BC)

$$\nabla u : X \to Y \tag{2}$$

and the convexity constraint(C)

$$u = convex \tag{3}$$

Finite Difference Methods

- Numerical techniques for solving differential equations
- Numerical solutions
- Derivation from Taylor's polynomial

$$u\left(x_1,\ldots,x_n\right)=u\left(x_1^0,\ldots,x_n^0\right)+\sum_{i=1}^{n}\left(x_i-x_i0\right)\frac{\partial u\left(x_1^0,\ldots,x_n^0\right)}{\partial x_i} \tag{4}$$

$$+\frac{1}{2!}\sum_{i,j=1}^{n}\left(x_i-x_i^0\right)\left(x_j-x_j^0\right)\frac{\partial u\left(x_1^0,\ldots,x_n^0\right)}{\partial x_i\partial x_j}+O(h^2) \tag{5}$$

Finite Difference Operators

$$\frac{\partial u}{\partial x_i} = \frac{u\left(\ldots,x_i^{m+1},\ldots\right)-u\left(\ldots,x_i^{m-1},\ldots\right)}{2h}+O\left(h^2\right) \tag{6}$$

$$\frac{\partial^2 u}{\partial x_i^2} = \frac{u\left(,x_i^{m+1},\right)+u\left(,x_i^{m-1},\right)-2u\left(,x_i^{m},\right)}{h^2}+O\left(h^2\right) \tag{7}$$

Finite Difference Methods

$$u(x_1, \dots, x_n) = u\left(x_1^0, \dots, x_n^0\right) + \sum_{i=1}^{n} (x_i - x_i 0) \frac{\partial u\left(x_1^0, \dots, x_n^0\right)}{\partial x_i} \tag{8}$$

$$+\frac{1}{2!} \sum_{i,j=1}^{n} \left(x_i - x_i^0\right) \left(x_j - x_j^0\right) \frac{\partial u\left(x_1^0, \dots, x_n^0\right)}{\partial x_i \partial x_j} + O(h^2) \tag{9}$$

Finite Difference Operators

$$\frac{\partial u}{\partial x_i} = \frac{u\left(\dots, x_i^{m+1}, \dots\right) - u\left(\dots, x_i^{m-1}, \dots\right)}{2h} + O\left(h^2\right) \tag{10}$$

$$\frac{\partial^2 u}{\partial x_i^2} = \frac{u\left(, x_i^{m+1}, \right) + u\left(, x_i^{m-1}, \right) - 2u\left(, x_i^{m}, \right)}{h^2} + O\left(h^2\right) \tag{11}$$

$$\frac{\partial^2 u}{\partial x_i \partial x_j} = \frac{1}{2h^2}\left[u\left(x_i^{m+1}, x_j^{m+1}\right) + u\left(x_i^{m-1}, x_j^{m-1}\right)\right. \tag{12}$$

$$\left. -u\left(x_i^{m+1}, x_j^{m-1}\right) - u\left(x_i^{m-1}, x_j^{m+1}\right)\right] + O\left(h^2\right) \tag{13}$$

Finite Difference Methods

$$\frac{\partial u}{\partial x_i} = \frac{u\left(\dots, x_i^{m+1}, \dots\right) - u\left(\dots, x_i^{m-1}, \dots\right)}{2h} + O\left(h^2\right) \quad (14)$$

$$\frac{\partial^2 u}{\partial x_i^2} = \frac{u\left(, x_i^{m+1}, \right) + u\left(, x_i^{m-1}, \right) - 2u\left(, x_i^{m}, \right)}{h^2} + O\left(h^2\right) \quad (15)$$

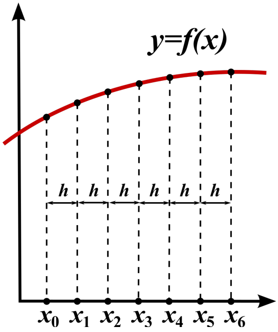

Figure 1: The finite difference method relies on discretizing a function on a grid

Finite Difference Methods

more than 100,000,000 points

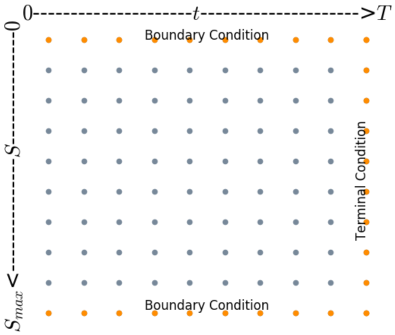

Figure 2: The finite difference method relies on discretizing a function on a grid

Simplification of MAPDE

Monge-Ampere Partial Differential Equation

$$\det\left(\frac{\partial^2 u}{\partial x_i \partial x_j}\right)(x) = \frac{\mu(x)}{\nu \circ \nabla u(x)} \tag{16}$$

$$\begin{bmatrix} \frac{\partial^2 u}{\partial x_1^2} & \frac{\partial^2 u}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 u}{\partial x_1 \partial x_n} \\ \frac{\partial^2 u}{\partial x_2 \partial x_1} & \frac{\partial^2 u}{\partial x_2^2} & \cdots & \frac{\partial^2 u}{\partial x_2 \partial x_n} \\ \vdots & & & \\ \frac{\partial^2 u}{\partial x_n \partial x_1} & \frac{\partial^2 u}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 u}{\partial x_n^2} \end{bmatrix} = \frac{\mu(x_1, x_2, \ldots, x_n)}{\nu\left(\frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}\right)} \tag{17}$$

Simplification of MAPDE

$$\det{}^{+}\left(D^2 u\right) = \min_{(\nu_1,\ldots,\nu_d)\in V} \left\{ \prod_{j=1}^{d} \max\left\{u_{\nu_j\nu_j}, 0\right\} + \sum_{j=1}^{d} \min\left\{u_{\nu_j\nu_j}, 0\right\} \right\} \quad (18)$$

Set $\mu, \nu = x_1 + x_2 + \ldots + x_D$, A standard centered difference discretization of this equation is

$$f\left(u^n\right) = \prod_{j=1}^{4} D_{v_j v_j} u + \sum_{j=1}^{4} D_{v_j v_j} u - \frac{x_1 + x_1 + x_3 + x_4}{D_{x_1} + D_{x_2} + D_{x_3} + D_{x_4}} \quad (19)$$

System of Equations

$$\prod_{j=1}^{4} C*(u_0^j + u_2^j - 2u_1^j) + \sum_{j=1}^{4} C*(u_0^j + u_2^j - 2u_1^j) = f(1) \quad (20)$$

$$\prod_{j=1}^{4} C*(u_1^j + u_3^j - 2u_2^j) + \sum_{j=1}^{4} C*(u_1^j + u_3^j - 2u_2^j) = f(2) \quad (21)$$

$$\prod_{j=1}^{4} C*(u_2^j + u_4^j - 2u_3^j) + \sum_{j=1}^{4} C*(u_2^j + u_4^j - 2u_3^j) = f(3) \quad (22)$$

$$\ldots \quad (23)$$

$$\ldots \quad (24)$$

$$\prod_{j=1}^{4} C*(u_{99}^j + u_{101}^j - 2u_{100}^j) + \sum_{j=1}^{4} C*(u_{99}^j + u_{101}^j - 2u_{100}^j) = f(100) \quad (25)$$

Newton Iteration

$$\frac{\partial f\left(u^n\right)}{\partial u_1'} = \prod_{j\neq 1}^{4} C * (u_1^j + u_3^j - 2u_2^j) + C + \frac{x_1 + x_2 + x_3 + x_4}{B * u_1^2} \quad (26)$$

$$\frac{\partial f\left(u^n\right)}{\partial u_3'} = \prod_{j\neq 1}^{4} C * (u_1^j + u_3^j - 2u_2^j) + C - \frac{x_1 + x_2 + x_3 + x_4}{B * u_3^2} \quad (27)$$

$$\frac{\partial f\left(u^n\right)}{\partial u_2'} = \sum_{j=1}^{4}(-2) * \prod_{j\neq 1}^{4} C * (u_1^j + u_3^j - 2u_2^j) - 8C \quad (28)$$

$$u^{n+1} = u^n - \nabla F\left(u^n\right)^{-1} F\left(u^n\right) \quad (29)$$

Data Frame

```
SolutionMax = 10
SolutionMin = 0
Step_h = 1
D_length = int((SolutionMax - SolutionMin)/Step_h)
dataInit_Solution = np.random.rand(D_length+4,D_length
    +4,D_length+4,D_length+4)

def getData(U,D_length):
    data = []
    for x1 in range(D_length):
        for x2 in range(D_length):
            for x3 in range(D_length):
                for x4 in range(D_length):
                    data.append((x1,x2,x3,x4,U[x1:x1
                        +4,x2:x2+4,x3:x3+4,x4:x4+4]))

    return data
```

Loss Function

```python
def loss(step_h):
    C = 1/(step_h*step_h)
    B = 0.5/step_h

    return lambda x: C * (x[4][1,2,2,2] + x[4][3,2,2,2] - 2*x[4][2,2,2,2]) *\
                     C * (x[4][2,1,2,2] + x[4][2,3,2,2] - 2*x[4][2,2,2,2]) *\
                     C * (x[4][2,2,1,2] + x[4][2,2,3,2] - 2*x[4][2,2,2,2]) *\
                     C * (x[4][2,2,2,1] + x[4][2,2,2,3] - 2*x[4][2,2,2,2]) +\
                     C * (x[4][1,2,2,2] + x[4][3,2,2,2] - 2*x[4][2,2,2,2]) +\
                     C * (x[4][2,1,2,2] + x[4][2,3,2,2] - 2*x[4][2,2,2,2]) +\
                     C * (x[4][2,2,1,2] + x[4][2,2,3,2] - 2*x[4][2,2,2,2]) +\
                     C * (x[4][2,2,2,1] + x[4][2,2,2,3] - 2*x[4][2,2,2,2]) -\
                     (x[0]+x[1]+x[2]+x[3]+8)/(B*(-x[4][1,2,2,2] + x[4][3,2,2,2] \
                                                 -x[4][2,1,2,2] + x[4][2,3,2,2] \
                                                 -x[4][2,2,1,2] + x[4][2,2,3,2] \
                                                 -x[4][2,2,2,1] + x[4][2,2,2,3]))
```

Figure 3: The Loss function

```python
spark_U = sc.parallelize(getData(dataInit_Solution,
    D_length),40)
U_loss = spark_U.map(loss(Step_h)).reduce(lambda x,y:
    abs(x)+abs(y))
```

Finite Difference Operators

$$\frac{\partial f\left(u^{n}\right)}{\partial u_{1}^{\prime}} = \prod_{j\neq 1}^{4} C*(u_1^j+u_3^j-2u_2^j)+C+\frac{x_1+x_2+x_3+x_4}{B*u_1^2} \quad (30)$$

```python
def Dv1(step_h,Dimension):
    C = 1/(step_h*step_h)
    B = 0.5/step_h

    return lambda x:((x[0],x[1],x[2],x[3]),C *\
                     C * (x[4][3,1,2,2] + x[4][3,3,2,2] - 2*x[4][3,2,2,2]) *\
                     C * (x[4][3,2,1,2] + x[4][3,2,3,2] - 2*x[4][3,2,2,2]) *\
                     C * (x[4][3,2,2,1] + x[4][3,2,2,3] - 2*x[4][3,2,2,2]) + C -\
                     (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][3,2,2,2]*x[4][3,2,2,2]) +\
                     C * (x[4][1,3,2,2] + x[4][3,3,2,2] - 2*x[4][2,3,2,2]) *\
                     C *\
                     C * (x[4][2,3,1,2] + x[4][2,3,3,2] - 2*x[4][2,3,2,2]) *\
                     C * (x[4][2,3,2,1] + x[4][2,3,2,3] - 2*x[4][2,3,2,2]) + C -\
                     (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][2,3,2,2]*x[4][2,3,2,2]) +\
                     C * (x[4][1,2,3,2] + x[4][3,2,3,2] - 2*x[4][2,2,3,2]) *\
                     C * (x[4][2,1,3,2] + x[4][2,3,3,2] - 2*x[4][2,2,3,2]) *\
                     C *\
                     C * (x[4][2,2,3,1] + x[4][2,2,3,3] - 2*x[4][2,2,3,2]) + C -\
                     (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][2,2,3,2]*x[4][2,2,3,2]) +\
                     C * (x[4][1,2,2,3] + x[4][3,2,2,3] - 2*x[4][2,2,2,3]) *\
                     C * (x[4][2,1,2,3] + x[4][2,3,2,3] - 2*x[4][2,2,2,3]) *\
                     C * (x[4][2,2,1,3] + x[4][2,2,3,3] - 2*x[4][2,2,2,3]) *\
                     C + C -\
                     (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][2,2,2,3]*x[4][2,2,2,3]))
```

Finite Difference Operators

$$\frac{\partial f\left(u^n\right)}{\partial u_2'} = \sum_{j=1}^{4}(-2) * \prod_{j\neq 1}^{4} C * (u_1^j + u_3^j - 2u_2^j) - 8C \qquad (31)$$

```python
def Dv2(step_h,Dimension):
    C = 1/(step_h*step_h)

    return lambda x:((x[0],x[1],x[2],x[3]),
                        C * (-2) *\
                        C * (x[4][2,1,2,2] + x[4][2,3,2,2] - 2*x[4][2,2,2,2]) *\
                        C * (x[4][2,2,1,2] + x[4][2,2,3,2] - 2*x[4][2,2,2,2]) *\
                        C * (x[4][2,2,2,1] + x[4][2,2,2,3] - 2*x[4][2,2,2,2]) +\
                        C * (x[4][1,2,2,2] + x[4][3,2,2,2] - 2*x[4][2,2,2,2]) *\
                        C * (-2) *\
                        C * (x[4][2,2,1,2] + x[4][2,2,3,2] - 2*x[4][2,2,2,2]) *\
                        C * (x[4][2,2,2,1] + x[4][2,2,2,3] - 2*x[4][2,2,2,2]) +\
                        C * (x[4][1,2,2,2] + x[4][3,2,2,2] - 2*x[4][2,2,2,2]) *\
                        C * (x[4][2,1,2,2] + x[4][2,3,2,2] - 2*x[4][2,2,2,2]) *\
                        C * (-2) *\
                        C * (x[4][2,2,2,1] + x[4][2,2,2,3] - 2*x[4][2,2,2,2]) +\
                        C * (x[4][1,2,2,2] + x[4][3,2,2,2] - 2*x[4][2,2,2,2]) *\
                        C * (x[4][2,1,2,2] + x[4][2,3,2,2] - 2*x[4][2,2,2,2]) *\
                        C * (x[4][2,2,1,2] + x[4][2,2,3,2] - 2*x[4][2,2,2,2]) *\
                        C * (-2) - 2*C*Dimension)
```

Figure 5: $\frac{\partial f(u^n)}{\partial u_2'}$

Finite Difference Operators

$$\frac{\partial f\left(u^n\right)}{\partial u_3^{\prime}}=\prod_{j \neq 1}^4 C *\left(u_1^j+u_3^j-2 u_2^j\right)+C-\frac{x_1+x_2+x_3+x_4}{B * u_3^2} \quad (32)$$

```
def Dv3(step_h,Dimension):
    C = 1/(step_h*step_h)
    B = 0.5/step_h

    return lambda x:((x[0],x[1],x[2],x[3]),C *\
                    C * (x[4][1,1,2,2] + x[4][1,3,2,2] - 2*x[4][1,2,2,2]) *\
                    C * (x[4][1,2,1,2] + x[4][1,2,3,2] - 2*x[4][1,2,2,2]) *\
                    C * (x[4][1,2,2,1] + x[4][1,2,2,3] - 2*x[4][1,2,2,2]) + C +\
                    (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][1,2,2,2]*x[4][1,2,2,2]) +\
                    C * (x[4][1,1,2,2] + x[4][3,1,2,2] - 2*x[4][2,1,2,2]) *\
                    C *\
                    C * (x[4][2,1,1,2] + x[4][2,1,3,2] - 2*x[4][2,1,2,2]) *\
                    C * (x[4][2,1,2,1] + x[4][2,1,2,3] - 2*x[4][2,1,2,2]) + C +\
                    (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][2,1,2,2]*x[4][2,1,2,2]) +\
                    C * (x[4][1,2,1,2] + x[4][3,2,1,2] - 2*x[4][2,2,1,2]) *\
                    C * (x[4][2,1,1,2] + x[4][2,3,1,2] - 2*x[4][2,2,1,2]) *\
                    C *\
                    C * (x[4][2,2,1,1] + x[4][2,2,1,3] - 2*x[4][2,2,1,2]) + C +\
                    (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][2,2,1,2]*x[4][2,2,1,2]) +\
                    C * (x[4][1,2,2,1] + x[4][3,2,2,1] - 2*x[4][2,2,2,1]) *\
                    C * (x[4][2,1,2,1] + x[4][2,3,2,1] - 2*x[4][2,2,2,1]) *\
                    C * (x[4][2,2,1,1] + x[4][2,2,3,1] - 2*x[4][2,2,2,1]) *\
                    C + C +\
                    (x[0]+x[1]+x[2]+x[3]+8)/(B*x[4][2,2,2,1]*x[4][2,2,2,1]))
```

Newton Iteration

```
Drive2 = spark_U.map(Dv2(Step_h,Dimension))
Drive1 = spark_U.map(Dv1(Step_h,Dimension))
Drive3 = spark_U.map(Dv3(Step_h,Dimension))
AllDrive = Drive2.join(Drive3).mapValues(lambda x: x
    [0]+x[1]).join(Drive1).mapValues(lambda x: x[0]+x
    [1])
AllDrive = AllDrive.sortBy(lambda x:x[0]).values().
    collect()

for x1 in range(D_length):
  for x2 in range(D_length):
    for x3 in range(D_length):
      for x4 in range(D_length):
        index = x1 *D_length*D_length*D_length + x2*
            D_length*D_length + x3 *D_length + x4
        dataInit_Solution[x1+2,x2+2,x3+2,x4+2] =
            dataInit_Solution[x1+2,x2+2,x3+2,x4+2] -
            rate*AllDrive[index]
```

Plan

- $F\left(u^{n}\right)=0, Dimension=2$ (By Xinyue Wang)
- $F\left(u^{n}\right)=\frac{\rho_X(x)}{\rho_Y(\nabla u(x))}, Dimension=2$(By Qiming Yuan)
- $F\left(u^{n}\right)=0, Dimension=4$ (By Mingjia Xue)
- $F\left(u^{n}\right)=\frac{\rho_X(x)}{\rho_Y(\nabla u(x))}, Dimension=4$(By Tianma Shen)

Experiment1

$F\left(u^n = 0, Dimension = 2\right.$


```
Iter: 0 PDE Loss: 10852.344610320672 11.54680315295904
Iter: 2 PDE Loss: 8260.100007689807 7.099948755523584
Iter: 4 PDE Loss: 5992.057312727652 3.341697263047692
Iter: 6 PDE Loss: 4319.849146449821 1.6299627831047665
Iter: 8 PDE Loss: 3386.725645498688 0.826088842547815
Iter: 10 PDE Loss: 3040.615116353076 0.4647440808251315
Iter: 12 PDE Loss: 2976.4141058345717 0.42560341252064804
Iter: 14 PDE Loss: 2915.3305765199025 0.3916841916062008
Iter: 16 PDE Loss: 2857.5158223832314 0.35873759239688807
Iter: 18 PDE Loss: 2802.2367024547034 0.3269935670501507
Iter: 20 PDE Loss: 2749.7450754471147 0.2965728363358924
Iter: 22 PDE Loss: 2699.448767633719 0.26749817981660673
Iter: 24 PDE Loss: 2651.4664388129913 0.23981082717976054
Iter: 26 PDE Loss: 2605.416482067203 0.21356751614078506
Iter: 28 PDE Loss: 2561.334840606903 0.19350089093056155
Iter: 30 PDE Loss: 2519.607471130129 0.1747754070054781
Iter: 32 PDE Loss: 2479.9826681627096 0.1571894494570718
Iter: 34 PDE Loss: 2443.0773113394926 0.1407301721827614
Iter: 36 PDE Loss: 2409.0507101981407 0.12539642217632752
Iter: 38 PDE Loss: 2377.801750552403 0.11136442343203701
Iter: 40 PDE Loss: 2349.843533355279 0.1018754132746964
Iter: 42 PDE Loss: 2335.970146659389 0.09671092845230111
Iter: 44 PDE Loss: 2333.5339437225707 0.09578112455466381
Iter: 46 PDE Loss: 2331.149752493049 0.09485333290784892
Iter: 48 PDE Loss: 2328.805132635027 0.09392761081424994
Iter: 50 PDE Loss: 2326.5111435233152 0.09300401507453171
Iter: 52 PDE Loss: 2324.2467291281555 0.09208260197464946
Iter: 54 PDE Loss: 2322.028374466761 0.091163427272831
Iter: 56 PDE Loss: 2319.849453290693 0.0902465461865356
Iter: 58 PDE Loss: 2317.7242029370764 0.08933201337942531
Iter: 60 PDE Loss: 2315.6241423564206 0.08841988294836511
Iter: 62 PDE Loss: 2313.554447458701 0.0875102084104753
Iter: 64 PDE Loss: 2311.4959853196506 0.08660304269027108
Iter: 66 PDE Loss: 2309.45629968595 0.08569843810690125
Iter: 68 PDE Loss: 2307.4408061954164 0.08479644636152028
Iter: 70 PDE Loss: 2305.475123298784 0.08389711852481652
Iter: 72 PDE Loss: 2303.5754749415805 0.08300050502472356
Iter: 74 PDE Loss: 2301.7287838864086 0.08210665563434372
Iter: 76 PDE Loss: 2299.9102768143593 0.08121561946010325
Iter: 78 PDE Loss: 2298.117917092782 0.08032744493017452
```

Figure 7: $F\left(u^n\right) = 0, Dimension = 2$

Experiment2

$$F\left(u^n\right) = \frac{\rho_X(x)}{\rho_Y(\nabla u(x))}, Dimension = 2$$

exp1_100x100_2

```
Iter:   0 PDE Loss: 398591.04582390847 125.89330097840815
Iter:  20 PDE Loss: 230883.97215962244 74.27679328617702
Iter:  40 PDE Loss: 157569.73185035205 47.34405457491104
Iter:  60 PDE Loss: 122170.34187689456 36.141954172377424
Iter:  80 PDE Loss: 102860.58452522026 28.474702509488715
Iter: 100 PDE Loss:  90917.7178506266  23.91086755454451
Iter: 120 PDE Loss:  82843.25966802661 20.28626197653224
Iter: 140 PDE Loss:  76966.86927685936 17.361364902689754
Iter: 160 PDE Loss:  72410.55469278751 14.983819838241622
Iter: 180 PDE Loss:  68732.15266858094 13.036301491457978
Iter: 200 PDE Loss:  65683.40606955494 11.428403058978876
Iter: 220 PDE Loss:  63127.89754308999 10.090330751887558
Iter: 240 PDE Loss:  60939.94379311528 9.185695431363035
Iter: 260 PDE Loss:  59036.09034991049 8.506861476008794
Iter: 280 PDE Loss:  57353.5878941759  7.884322062567303
Iter: 300 PDE Loss:  55850.63201441553 7.315651523575696
Iter: 320 PDE Loss:  54494.01787864732 6.888490283035016
Iter: 340 PDE Loss:  53262.55013271723 6.544495090601078
Iter: 360 PDE Loss:  52141.62332523391 6.226664566893819
Iter: 380 PDE Loss:  51121.25410970803 5.9328384372770735
Iter: 400 PDE Loss:  50183.68499502943 5.660896162479165
Iter: 420 PDE Loss:  49320.09357916603 5.408831187586488
Iter: 440 PDE Loss:  48527.64742880279 5.174787512584769
Iter: 460 PDE Loss:  47788.26078338492 4.957073290615988
Iter: 480 PDE Loss:  47097.16344193429 4.754160841700786
Iter: 500 PDE Loss:  46446.17141434873 4.56467902553861
Iter: 520 PDE Loss:  45835.643108799   4.387401691035844
Iter: 540 PDE Loss:  45262.52852123505 4.221234486200802
Iter: 560 PDE Loss:  44719.31613737693 4.065201392872272
Iter: 580 PDE Loss:  44205.6079307285  3.9184317660607757
Iter: 600 PDE Loss:  43721.10006143142 3.7801482899634866
Iter: 620 PDE Loss:  43261.14938425883 3.649656035333029
Iter: 640 PDE Loss:  42823.36799433775 3.5263326658396927
Iter: 660 PDE Loss:  42407.67419360704 3.4096197617598136
Iter: 680 PDE Loss:  42010.32647531223 3.299015186512131
```

Figure 8: $F\left(u^n\right) = \frac{\rho_X(x)}{\rho_Y(\nabla u(x))}, Dimension = 2$